Mobile Pentium® III Processor Specification Update

Release Date: December 1999

Order Number: 245306-002

The Intel® mobile Pentium® III processor or the Intel® Pentium® III Processor Mobile Module may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY....................ii
PREFACE....................iii
Nomenclature....................iii

Mobile Pentium® III Processor Specification Update

GENERAL INFORMATION....................2
Intel® Mobile Pentium® III Processor (Micro-PGA2) Markings....................2
Intel® Mobile Pentium® III Processor (BGA2) Markings....................2
Identification Information....................4
Summary of Changes....................5
Codes Used in Summary Table....................5
Summary of Errata....................6
Summary of Documentation Changes....................7
Summary of Specification Clarifications....................8
ERRATA....................9
DOCUMENTATION CHANGES....................28
SPECIFICATION CLARIFICATIONS....................30
SPECIFICATION CHANGES....................32

REVISION HISTORY

Date of Revision	Version	Description
October 1999	-001	Initial release
December 1999	-002	Added Errata K45, K46, K47, and K2MO; added Documentation Change K2; added Specification Clarification K1; added Specification Change K1; Updated the reference to the published datasheets in the Preface; updated the *Mobile Pentium III Processor in BGA2 and micro-PGA2 Packages Identification Information* table.

PREFACE

Please note that both the mobile Pentium® III processor in BGA2 and Micro-PGA2 packages and the Intel® Mobile Pentium® III Processor Mobile Module are unannounced products.

This document is an update to the specifications contained in the following documents:

- *Mobile Pentium® III Processor in BGA2 and Micro-PGA2 Packages at 400 MHz, 450 MHz, and 500 MHz* datasheet (Order Number 24530201)
- *Pentium® III Processor Mobile Module MMC2* datasheet (Order Number 24530401*)*
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3* (Order Numbers 243190, 243191, and 243192, respectively).

This document is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Errata, Documentation Changes, Specification Clarifications, and Specification Changes.

Nomenclature

S-Spec Number is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. Errata may cause the processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Documentation Changes include errors (including typographical), or omissions from the current published specifications. These changes will be incorporated in the next release of the appropriate documentation(s).

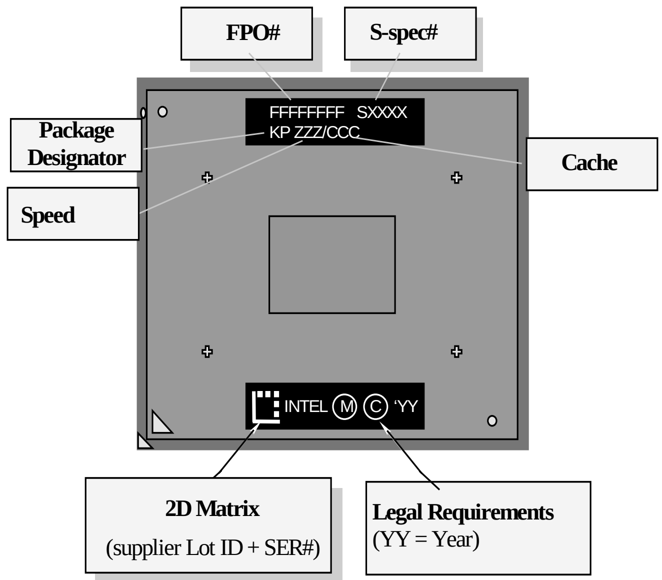
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the appropriate documentation(s).

Specification Changes are modifications to the current published specifications for the processor. These changes will be incorporated in the next release of the appropriate documentation(s).

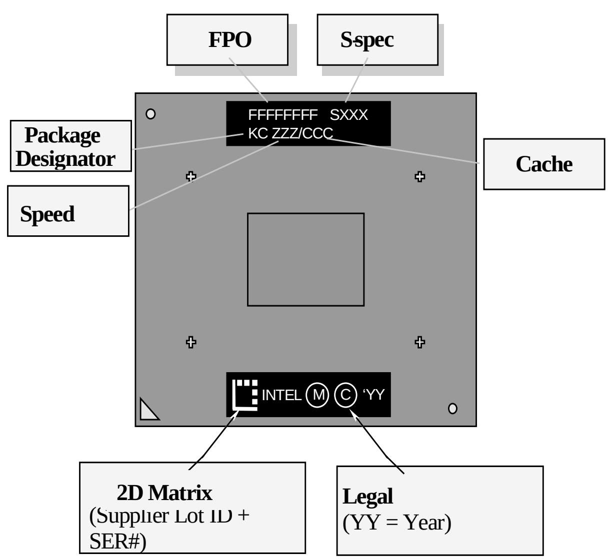
Mobile Pentium® III Processor Specification Update

GENERAL INFORMATION

Intel Mobile Pentium III Processor (Micro-PGA2) Markings



Intel Mobile Pentium III Processor (BGA2) Markings



Intel Pentium III Processor Mobile Module Markings

The Product Tracking Code (PTC) determines the Intel assembly level of the module. The PTC is on the secondary side of the module and provides the following information:

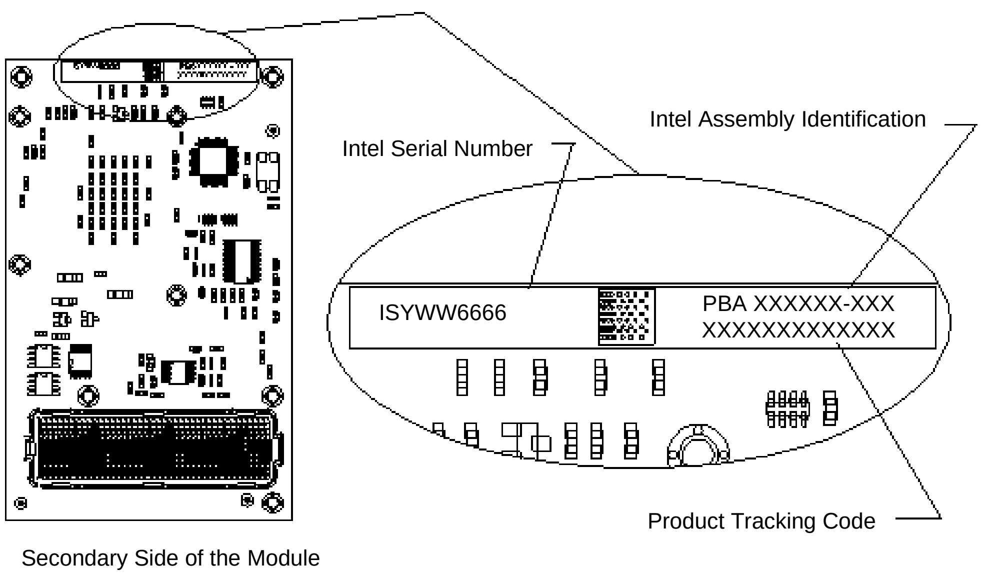
Example: ***PML50002001AA***

- The PTC will consist of 13 characters as identified in the above example and can be broken down as follows:

AABCCCDDEEEFF

- Definition:

	AA	-	Processor Module = PM
	B	-	Intel® Pentium® III Processor Mobile Module = L
	CCC	-	Speed Identity = 450 or 500
	DD	-	Cache Size = 02 (256 KB)
	EEE	-	Notifiable Design Revision (Start at 001)
	FF	-	Notifiable Processor Revision (Start at AA)



IDENTIFICATION INFORMATION

The mobile Pentium III processor can be identified by the following values:

Family[1]	Model[2]	Brand ID[3]
0110	1000	00000010

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after Reset, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after Reset, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.
3. The Brand ID is returned by the CPUID instruction in the EBX[7:0] when CPUID is executed with the value of 1 in the EAX.

Intel Mobile Pentium III Processor in BGA2 and micro-PGA2 Packages Identification Information

S-Spec	Product Stepping	CPUID	Speed (MHz) Core/Bus	Integrated L2 Size (Kbytes)	Package	Notes
SL3DU	BA2	0681h	400/100	256	BGA	1
SL3KX	BA2	0681h	450/100	256	BGA	2
SL3DT	BA2	0681h	500/100	256	BGA	2
SL3LG	PA2	0681h	450/100	256	Micro-PGA	2
SL3DW	PA2	0681h	500/100	256	Micro-PGA	2

NOTES:

1. **V_{CC_CORE} = 1.35V**
2. V_{CC_CORE} = 1.60V

Intel Pentium III Processor Mobile Module Identification Information

PTC	Product Stepping	CPUID	Speed (MHz) Core/Bus	Integrated L2 Size (Kbytes)	Package	Notes
PML45002001AA	MA2	0681h	450/100	256	MMC2	1
PML50002001AA	MA2	0681h	500/100	256	MMC2	1

NOTES:

1. Vcore voltage is 1.6 V.

SUMMARY OF CHANGES

The following table indicates the Errata, Documentation Changes, Specification Clarifications, or Specification Changes that apply to mobile Pentium III processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Doc:	Intel intends to update the appropriate documentation in a future revision.
AP:	APIC related erratum.
MO:	Mobile processor only erratum
PKG:	This column refers to errata on the mobile Pentium III processor substrate.
Shaded:	This item is either new or modified from the previous version of the document.

Each Specification Update item will be prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Pentium® II processor

B = Intel® Mobile Pentium® II processor

C = Intel® Celeron™ processor

D = Pentium® II Xeon™ processor

E = Pentium® III processor

G = Pentium® III Xeon™ processor

H = Intel® Mobile Celeron™ processor

I = Pentium® III processor

J = Pentium® III Xeon™ processor

K = Mobile Pentium® III processor

The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products do not use this convention.

Summary of Errata

NO.	BA2	PA2	mA2	PKG	Plans	ERRATA
K1	X	X	X		NoFix	FP data operand pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
K2	X	X	X		NoFix	Differences exist in debug exception reporting
K3	X	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
K4	X	X	X		NoFix	Double ECC error on read may result in BINIT#
K5	X	X	X		NoFix	FP inexact-result exception flag may not be set
K6	X	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
K7	X	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
K8	X	X	X		NoFix	Branch traps do not function if BTMs are also enabled
K9	X	X	X		NoFix	Machine check exception handler may not always execute successfully
K10	X	X	X		NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
K11	X	X	X		NoFix	LBER may be corrupted after some events
K12	X	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
K13	X	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
K14	X	X	X		No Fix	Memory type undefined for non-memory operations
K15	X	X	X		NoFix	FP Data operand pointer may not be zero after power on or Reset
K16	X	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
K17	X	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
K18	X	X	X		NoFix	Read portion of RMW instruction may execute twice
K19	X	X	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
K20	X	X	X		NoFix	MOV with debug register causes debug exception
K21	X	X	X		NoFix	Upper four PAT entries not usable with Mode B or Mode C paging
K22	X	X	X		NoFix	Data breakpoint exception in a displacement relative near call may corrupt EIP
K23	X	X	X		NoFix	RDMSR and WRMSR to invalid MSR may not cause GP fault
K24	X	X	X		NoFix	SYSENTER/SYSEXIT instructions can implicitly load null segment selector to SS and CS registers
K25	X	X	X		NoFix	PRELOAD followed by EXTEST does not load boundary scan data
K26	X	X	X		NoFix	INT 1 instruction handler execution could generate a debug exception
K27	X	X	X		NoFix	Misaligned Locked access to APIC space results in a hang
K28	X	X	X		NoFix	Processor may assert DRDY# on a write with no data.
K29	X	X	X		NoFix	GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6
K30	X	X	X		NoFix	Machine check exception may occur due to improper line eviction in the IFU

Summary of Errata

NO.	BA2	PA2	mA2	PKG	Plans	ERRATA
K31	X	X	X		NoFix	Performance counters include streaming SIMD extensions L1 prefetch
K32	X	X	X		NoFix	Processor will erroneously report a BIST failure
K33					Fix	Internal snooping mechanism causes livelock condition
K34	X	X	X		NoFix	Cache coherency may be lost if snoop occurs during cache line invalidation
K35					Fix	Extra DRDY# assertion when eviction back-to-back write combining lines
K36	X	X	X		NoFix	ECC detection and correction issue
K37	X	X	X		NoFix	L2_LD and L2_M_LINES_OUTM performance-monitoring counters do not work
K38	X	X	X		NoFix	Snoop request may cause DBSY# hang
K39	X	X	X		NoFix	IFU/DCU deadlock may cause system hang
K40					Fix	WBINVD may lock write out buffer
K41	X	X	X		NoFix	L2_DBUS_BUSY performance monitoring counter will not count writes
K42	X	X	X		NoFix	Lower bits of SMRAM SMBASE register cannot be written with an ITP
K43	X	X	X		NoFix	Task switch caused by page fault may cause wrong PTE and PDE accessed bit to be set
K44	X	X	X		NoFix	Cross-Modifying code operations on a jump instruction may cause general protection fault
K45	X	X	X		NoFix	Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination
K46	X	X	X		NoFix	MASKMOVQ Instruction Interaction with String Operation May Cause Deadlock
K47	X	X	X		Fix	Noise Sensitivity Issue on Processor SMI# Pin
K1MO	X	X	X		Fix	Spurious interprocessor and end-of-interrupt message generated on the APIC bus in the quick start state
K2MO	X	X	X		Fix	APIC Has Unpredictable Interactions With the Quick Start State
K1AP	X	X	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
K2AP	X	X	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

Summary of Documentation Changes

NO.	BA2	PA2	MA2	PKG	Plans	Documentation Changes
K1	X	X	X		Doc	Handling of Self-Modifying and Cross-Modifying Code
K2	X	X	X		Doc	Machine Check Architecture Initialization of MCi_STATUS Registers

Summary of Specification Clarifications

NO.	BA2	PA2	MA2	PKG	Plans	Specification Clarifications
K1	X	X			Doc	Output low Current I_{OL} Specification

Summary of Specification Changes

NO.	BA2	PA2	MA2	PKG	Plans	Specification Changes
K1	X	X			Doc	$I_{CCT, DSLP}$ Specification

ERRATA

K1. WBINVD May Lock Write Out Buffer

Problem: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

Implication: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

Workaround: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K2. Differences Exist in Debug Exception Reporting

Problem: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the mobile Pentium III processor, as described below:

Case 1: The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The mobile Pentium III processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

Case 2: In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

Case 3: If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

Case 4: If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

Case 5: When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

Implication: When debugging or when developing debuggers for a mobile Pentium III processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

Workaround: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K3. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

Problem: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

Implication: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

Workaround: The debug handler should clear breakpoint registers before they become disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K4. Double ECC Error on Read May Result in BINIT#

Problem: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the mobile Pentium III processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

Implication: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

Workaround: Though the ability to drive BINIT# can be disabled in the mobile Pentium III processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K5. FP Inexact-Result Exception Flag May Not Be Set

Problem: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

Implication: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a “sticky” bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

Workaround: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K6. BTM for SMI Will Contain Incorrect FROM EIP

Problem: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

Implication: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K7. I/O Restart in SMM May Fail After Simultaneous MCE

Problem: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the mobile Pentium III processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

Implication: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

Workaround: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K8. Branch Traps Do Not Function If BTMs Are Also Enabled

Problem: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

Implication: The branch traps and branch trace message debugging features cannot be used together.

Workaround: If branch trap functionality is desired, BTMs must be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K9. Machine Check Exception Handler May Not Always Execute Successfully

Problem: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

Implication: No workaround which would guarantee successful MCE handler execution under this condition has been identified.

Workaround: If branch trap functionality is desired, BTMs must be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K10. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

Problem: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the mobile Pentium III processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

Implication: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K11. LBER May Be Corrupted After Some Events

Problem: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

Implication: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K12. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

Problem: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

Implication: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K13. Near CALL to ESP Creates Unexpected EIP Address

Problem: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

Implication: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

Workaround: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K14. Memory Type Undefined for Non-memory Operations

Problem: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

Implication: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

Workaround: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K15. FP Data Operand Pointer May Not Be Zero After Power On or Reset

Problem: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

Implication: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting in incorrect behavior of the software.

Workaround: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K16. MOVD Following Zeroing Instruction Can Cause Incorrect Result

Problem: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
or SUB EAX, EAX
2. MOVSX AX, BL
or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
or IMUL AX, BX, 1024 (opcode 69 /r iw)
or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

Implication: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

Workaround: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD, IMUL-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K17. Premature Execution of a Load Operation Prior to Exception Handler Invocation

Problem: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation,
2. If a waiting floating-point instruction or MMX instruction that performs a memory load has a floating-point exception pending, or
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances occur it is possible that the load portion of the instruction will have executed before the exception handler is entered.

Implication: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

Workaround: Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K18. Read Portion of RMW Instruction May Execute Twice

Problem: When the mobile Pentium III processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

Implication: This erratum has no effect if the memory targeted for the RMW instruction has no side-effects. If, however, the load targets a memory region that has side-effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

Workaround: Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then, the memory location will simply be read twice with no additional implications.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K19. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

Problem: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field, and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

Implication: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

Workaround: When decoding the MC2_STATUS MSR, reverse the two error fields.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K20. MOV With Debug Register Causes Debug Exception

Problem: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 14.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

Implication: With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

Workaround: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K21. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

Problem: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the mobile Pentium III processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

Implication: Only the lower four PAT entries are useful for 4-KB translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K22. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

Problem: If a data breakpoint is programmed at the memory location where the stack push of a near call is performed, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

Implication: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

Workaround: Do not program a data breakpoint exception on the stack where the push for the near call is performed.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K23. RDMSR or WRMSR to Invalid MSR Address May Not Cause GP Fault

Problem: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

Implication: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

Workaround: Do not use invalid MSR addresses with RDMSR or WRMSR.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K24. SYSENTER/SYSEXIT Instructions Can Implicitly Load “Null Segment Selector” to SS and CS Registers

Problem: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

Implication: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor’s behavior may become unpredictable, possibly resulting in system failure.

Workaround: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K25. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

Problem: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data “typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction.” As a result of this erratum, this method cannot be used to load the data onto the outputs.

Implication: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K26. INT 1 instruction handler execution could generate a debug exception

Problem: If the processor’s general detect enable flag is set and an explicit call is made to the interrupt procedure via the INT 1 instruction, the general detect enable flag should be cleared prior to entering the handler. As a result of this erratum, the flag is not cleared prior to entering the handler. If an access is made to the debug registers while inside of the handler, the state of the general detect enable flag will cause a second debug exception to be taken. The second debug exception clears the general detect enable flag and returns control to the handler which is now able to access the debug registers.

Implication: This erratum will generate an unexpected debug exception upon accessing the debug registers while inside of the INT 1 handler.

Workaround: Ignore the second debug exception that is taken as a result of this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K27. Misaligned Locked Access to APIC Space Results in Hang

Problem: When the processor’s APIC space is accessed with a misaligned locked access a machine check exception is expected. However, the processor’s machine check architecture is unable to handle the misaligned locked access.

Implication: If this erratum occurs the processor will hang. Typical usage models for the APIC address space do not use locked accesses. This erratum will not affect systems using such a model.

Workaround: Ensure that all accesses to APIC space are aligned.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K28. Processor May Assert DRDY# on a Write with No Data

Problem: When a MASKMOVQ instruction is misaligned across a chunk boundary in a way that one chunk has a mask of all 0’s, the processor will initiate two partial write transactions with one having all byte enables deasserted. Under these conditions, the expected behavior of the processor would be to perform both write transactions, but to deassert DRDY# during the transaction which has no byte enables asserted. As a result of this erratum, DRDY# is asserted even though no data is being transferred.

Implication: The implications of this erratum depend on the bus agent’s ability to handle this erroneous DRDY# assertion. If a bus agent cannot handle a DRDY# assertion in this situation, or attempts to use the invalid data on the bus during this transaction, unpredictable system behavior could result.

Workaround: A system which can accept a DRDY# assertion during a write with no data will not be affected by this erratum. In addition, this erratum will not occur if the MASKMOVQ is aligned.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K29. GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6

Problem: Writing a ‘1’ to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0h) will result in a general protection fault (GP#).

Implication: The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

Workaround: When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be ‘0.’ Implemented bits may be set as ‘0’ or ‘1’ as desired.

Status: For the steppings affected see the Summary of Changes at the beginning of this section.

K30. Machine Check Exception May Occur Due to Improper Line Eviction in the IFU

Problem: The mobile Pentium III processor is designed to signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism. Under a complex set of circumstances involving multiple speculative branches and memory accesses there exists a one cycle long window in which the processor may signal a MCE in the Instruction Fetch Unit (IFU) because instructions previously decoded have been evicted from the IFU. The one cycle long window is opened when an opportunistic fetch receives a partial hit on a previously executed but not as yet completed store resident in the store buffer. The resulting partial hit erroneously causes the eviction of a line from the IFU at a time when the processor is expecting the line to still be present. If the MCE for this particular IFU event is disabled, execution will continue normally.

Implication: While this erratum may occur on a system with any number of mobile Pentium III processors, the probability of occurrence increases with the number of processors. If this erratum does occur, a machine check exception will result. Note systems that implement an operating system that does not enable the Machine Check Architecture will be completely unaffected by this erratum (e.g., Windows95* and Windows98*).

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K31. Performance Counter L2 Prefetch Count Includes Streaming SIMD Extensions L1 Prefetch

Problem: The processors allow the measurement of the frequency and duration of numerous different internal and bus related events (see the *Intel Architecture Software Developer’s Manual, Volume 3,* for more details). The Streaming SIMD Extension (SSE) architecture provides a mechanism to pre-load data into the L1 cache, bypassing the L2 cache. The number of these L1 pre-loads measured by the performance monitoring logic will incorrectly be included in the count of "L2_LINES_IN" (24H) events.

Implication: If application software is run which utilizes the SSE L1 prefetch feature, the count of "L2_LINES_IN" (24H) will read a value that is greater than the correct value.

Workaround: The correct value of this counter may be calculated by taking the value read for L2_LINES_IN (24H) and subtracting from it the value read for "EMON_KNI_PREF_MISS" (4BH, Unit Mask 00H).

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K32. Processor Will Erroneously Report a BIST Failure

Problem: If the processor performs BIST at power-up, the EAX register is normally cleared (0H) if the processor passes BIST. The processor will erroneously report a non-zero value (signaling a BIST failure) even if BIST passes.

Implication: The processor will incorrectly signal an error after BIST is performed.

Workaround: The system BIOS should ignore the BIST results in the EAX register.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K33. Internal Snooping Mechanism Causes Livelock Condition

Problem: Internal timings may align where the L2 cache snooping mechanism and the Instruction Fetch Unit snooping mechanism reject each other's requests to the Data Cache Unit. Both units will continue to retry but reject requests on every other clock, leading to a livelock condition.

Implication: The system will hang. If an external agent is snooping the processor's caches, the hang will appear as an infinite snoop stall.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K34. Cache Coherency May Be Lost If Snoop Occurs During Cache Line Invalidation

Problem: There exists a two cycle window during a cache line invalidation (due to a WBINVD instruction or FLUSH# pin assertion) during which a processor performing a snoop of that line will not see the line in the cache. In addition, when this erratum occurs, the processor invalidating the line will not write back the data in that line.

Implication: If this erratum occurs, cache coherency and data will be lost.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K35. Extra DRDY# Assertion When Eviction Back-to-Back Write Combining Lines

Problem: The processor has the ability to evict back-to-back lines in its write combining buffers. If the processor writes back data from L1 to L2 during a back-to-back write combining line eviction, the processor may assert an extra DRDY# on the system bus.

Implication: Data corruption (loss of data) may occur.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K36. Limitation on Cache Line ECC Detection and Correction

Problem: ECC can detect and correct up to four single-bit ECC errors per cache line. However, the processor will only detect and correct one single-bit ECC error per cache line. While all ECC errors will be detected, multiple single bit errors will be incorrectly reported as uncorrectable double bit errors, rather than correctable single bit errors.

Implication: The processor may report fewer single bit ECC errors and more double bit ECC errors than previous processors.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K37. L2_LD and L2_M_LINES_OUTM Performance-Monitoring Counter Does Not Work

Problem: The L2_LD (29h) Performance-Monitoring counter, used for counting the number of L2 cache data loads, does not work properly.

Implication: This counter will report incorrect data.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K38. Snoop Request May Cause DBSY# Hang

Problem: A small window of time exists in which a snoop request originating from a bus agent to a processor with one or more outstanding memory transactions may cause the processor to assert DBSY# without issuing a corresponding bus transaction, causing the processor to hang (livelock). The exact circumstances are complex, and include the relative timing of internal processor functions with the snoop request from a bus agent.

Implication: This erratum may occur on a system with any number of processors. However, the probability of occurrence increases with the number of processors. If this erratum does occur, the system will hang with DBSY# asserted. At this point, the system requires a hard reset.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K39. IFU/DCU Deadlock May Cause System Hang

Problem: An internal deadlock situation may occur in systems with multiple bus agents, with a failure signature such that a processor either asserts DBSY# without issuing the corresponding data, or fails to respond to a snoop request from another bus agent. Should this erratum occur, the affected processor ceases code execution and the system will hang.

The specific circumstances surrounding the occurrence of this erratum are:

1. A locked operation to the Data Cache Unit (DCU) is in process.
2. A snoop occurs, but cannot complete due to the ongoing locked operation.
3. The presence of the snoop prevents pending Instruction Fetch Unit (IFU) requests from completing.
4. The IFU requests are periodically restarted.

The continued IFU restart attempts create additional DCU snoops, which prevent the in-process locked operation from completing, keeping the DCU locked.

Implication: The system may hang.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K40. WBINVD May Lock Write Out Buffer

Problem: If a processor is performing a WBINVD operation on a modified line, that line is stored in the processor's Write Out Buffer (WOB) until it is written to main memory. If another bus agent (such as a processor or PCI device) in the system generates a snoop that results in a hit to a modified line that is in the processor's WOB, that line could become permanently locked in the WOB. In addition to being locked in the WOB, the processor will not respond to the initial or subsequent snoop requests to this line, and the line in the WOB is never written to memory.

Implication: In the event of this erratum, coherency may be lost, which may result in a system lockup or system instability.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K41. L2_DBUS_BUSY Performance Monitoring Counter Will Not Count Writes

Problem: The L2_DBUS_BUSY (22H) performance monitoring counter is intended to count the number of cycles during which the L2 data bus is in use. For some steppings of the processor, the L2_DBUS_BUSY counter will not be incremented during write cycles and therefore will only reflect the number of L2 data bus cycles resulting from cache reads.

Implication: The L2_DBUS_BUSY event counts only L2 read cycles.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K42. Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP

Problem: The System Management Base (SMBASE) register (7EF8H) stores the starting address of the System Management RAM (SMRAM). This register is used by the processor when it is in System Management Mode (SMM), and its contents serve as the memory base for code execution and data storage. The 32-bit SMBASE register can normally be programmed to any value. When programmed with an In-Target Probe (ITP), however, any attempt to set the lower 11 bits of SMBASE to anything other than zeros via the WRMSR instruction will cause the attempted write to fail.

Implication: When set via ITP, any attempt to relocate SMRAM space must be made with 2 KB alignment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K43. Task Switch Caused by Page Fault May Cause Wrong PTE and PDE Access Bit to Be Set

Problem: If an operating system services a page fault through a Task Gate, the access bit may be set for an incorrect page table/directory entry.

Implication: An operating system which uses a Task Gate for its page fault handler may encounter this erratum. The effect of the erratum depends on the alignment of the Task State Segment (TSS), and ranges from no anomalous behavior to unexpected errors. Intel is not aware of any commercial operating systems which use a Task Gate to handle page faults. Task gates used for other purposes (NMI, Machine Check, or Double Fault) do not cause this erratum.

Workaround: The operating system may alternately use an Interrupt Gate or a Call Gate rather than a Task Gate.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K44. Cross-Modifying Code Operations on a Jump Instruction May Cause General Protection Fault

Problem: The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called Cross-Modifying Code (XMC). Software using XMC to modify the offset of an execution transfer instruction i.e. Jump, Call etc., without a synchronizing instruction may cause a General Protection Fault when the offset splits a cache line boundary.

Implication: Any application creating a General Protection Fault would be terminated by the operating system.

Workaround: Programmers should use the cross modifying code synchronization algorithm as detailed in Volume 3 of the *Intel Architecture Software Developer's Manual*, Section 7.1.3, in order to avoid this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K45. Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination

Problem: Intel's 32-bit Instruction Set Architecture (ISA) utilizes most of the available op-code space; however some byte combinations remain undefined and are considered illegal instructions. Intel processors detect the attempted execution of illegal instructions and signal an exception. This exception is handled by the operating system and/or application software.

Under a complex set of internal and external conditions involving illegal instructions, a deadlock may occur within the processor. The necessary conditions for the deadlock involve:

1. The illegal instruction is executed.
2. Two page table walks occur within a narrow timing window coincident with the illegal instruction.

Implication: The illegal instructions involved in this erratum are unusual and invalid byte combinations that are not useful to application software or operating systems. These combinations are not normally generated in the course of software programming, nor are such sequences known by Intel to be generated in commercially available software and tools. Development tools (compilers, assemblers) do not generate this type of code sequence, and will normally flag such a sequence as an error. If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K46. MASKMOVQ Instruction Interaction with String Operation May Cause Deadlock

Problem: Under the following scenario, combined with a specific alignment of internal events, the processor may enter a deadlock condition:

1. A store operation completes, leaving a write-combining (WC) buffer partially filled.
2. The target of a subsequent MASKMOVQ instruction is split across a cache line.
3. The data in (2) above results in a hit to the data in the WC buffer in (1).

Implication: If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K47. Noise Sensitivity Issue on Processor SMI# Pin

Problem: Post silicon characterization has demonstrated a greater than expected sensitivity to noise on the processor's SMI# input, which may result in spurious SMI# interrupts.

Implication: BIOS/SMM code that is capable of handling spurious SMI events will report a spurious SMI#, but should not be negatively impacted by this erratum. Systems whose BIOS code cannot handle spurious SMI events may fail, resulting in a system hang or other anomalous behavior.

Spurious SMI# interrupts should be controlled on the system board regardless of BIOS implementation.

Workaround: Possible workarounds that may reduce or eliminate the occurrence of the spurious SMI include:

1. Use a lower effective pull-up resistance on the SMI# pin. This resistor must meet the specifications of the component driving the SMI# signal.
2. Externally condition the SMI# signal prior to providing it to the processor's SMI# pin.

These workarounds should be evaluated on a design-by-design basis.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K1MO. Spurious Interprocessor and End-of-Interrupt Message Generated on the APIC Bus in the Quick Start State

Problem: Software can issue an interprocessor interrupt or an end-of-interrupt message on the APIC serial bus. If the processor enters the Quick Start state before the APIC bus message completes then the processor will continue to issue interprocessor interrupt or end-of-interrupt messages until the processor exits the Quick Start state. When the processor leaves the Quick Start state the APIC may report that a checksum error has occurred while sending the interprocessor interrupt or end-of-interrupt message.

Implication: If an I/O APIC sends a level sensitive interrupt to a processor it may receive many end-of-interrupt messages instead of just one. The processor may signal an APIC bus checksum error even though none has occurred. If a processor sends an interprocessor interrupt then more than one interrupt may be sent. The processor may signal an APIC bus checksum error.

Workaround: Configure the processor to use the Stop Grant state instead of the Quick Start state or use an interrupt controller other than the APIC.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K2MO. APIC Has Unpredictable Interactions With the Quick Start State

Problem: The processor cannot properly handle APIC bus messages or local interrupts in the Quick Start state. The local APIC timer does not work properly in the Quick Start state. If the processor is driving a message on the APIC bus while it is entering the Quick Start state it will continuously reissue the message and possibly report an APIC bus checksum error on exit from Quick Start. APIC bus messages that are not driven by the processor will be ignored by the processor if it enters the Quick Start state while the message is in progress.

Implication: The operating system cannot rely on the APIC behaving according to specification if the Quick Start state is used. This erratum only affects mobile platforms using APIC.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

Problem: APIC operations which access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter shutdown after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the mobile Pentium III processor to enter shutdown.

Implication: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.

Workaround: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

K2AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

Problem: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the mobile Pentium III processor despite the attempt to mask it via the LVT.

Implication: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

Workaround: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to:

- *Mobile Pentium® III Processor in BGA2 and Micro-PGA2 Packages 400 MHz, 450 MHz, and 500 MHz datasheet*
- *Pentium® III Processor Mobile Module MMC2* datasheet
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*

K1. Handling of Self-Modifying and Cross-Modifying Code

Section 7.1.3 paragraph 1. of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **should** be chosen.

It should state:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **must** be chosen.

Section 7.1.3 paragraph 6. of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code**. As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **should** be implemented.

It should state:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code.** As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **must** be implemented.

K2. Machine Check Architecture Initialization of MCi_STATUS Registers

Section 12.5, the last paragraph of the *Intel Architecture Software Developer's Manual Vol. 3: System Programming Guide* incorrectly states:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. This procedure is not included in the initialization pseudocode in Example 12-1.

It should state:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. <u>Following power cycling, the MCi_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all zeroes by software</u>. This procedure is not included in the initialization pseudocode in Example 12-1.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to:

- *Mobile Pentium® III Processor in BGA2 and Micro-PGA2 Packages 400 MHz, 450 MHz, and 500 MHz* datasheet
- *Pentium® III Processor Mobile Module MMC2*
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*

K1. Output Low Current I_{OL} Specification

This is a clarification to the *Mobile Pentium® III Processor in BGA2 and Micro-PGA2 Packages at 400 MHz, 450 MHz, and 500 MHz* datasheet (Order Number 24530201). Table 12 Should appear as the following.

Table 12. Clock, APIC, TAP, CMOS, and Open-drain Signal Group DC Specifications

T_{CASE} = 0°C to 100°C; V_{CC} = 1.35V ±100 mV or 1.6V ±115 mV; V_{CCT} = 1.5V ±115 mV					
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Notes**
V_{IL15}	Input Low Voltage, 1.5V CMOS	–0.15	$CMOSREF_{min}$ – 200 mV	V	
V_{IL25}	Input Low Voltage, 2.5V CMOS	–0.3	0.7	V	Notes 1, 2
$V_{IL,BCLK}$	Input Low Voltage, BCLK	–0.3	0.7	V	Note 2
V_{IH15}	Input High Voltage, 1.5V CMOS	$CMOSREF_{max}$ + 200 mV	V_{CCT}	V	
V_{IH25}	Input High Voltage, 2.5V CMOS	2.0	2.625	V	Notes 1, 2
$V_{IH,BCLK}$	Input High Voltage, BCLK	1.7	2.625	V	Note 2
V_{OL}	Output Low Voltage		0.4	V	Note 3
V_{OH15}	Output High Voltage, 1.5V CMOS	N/A	1.615	V	All outputs are Open-drain
V_{OH25}	Output High Voltage, 2.5V CMOS	N/A	2.625	V	All outputs are Open-drain
$V_{OH,VID}$	Output High Voltage, VID ball/pins	N/A	5.50	V	5V + 10%
$V_{CMOSREF}$	CMOSREF Voltage	0.90	1.10	V	Note 4
V_{CLKREF}	CLKREF Voltage	1.175	1.325	V	1.25V ±6% [4]
I_{OL}	Output Low Current	10		mA	Note 6
I_L	Leakage Current for Inputs, Outputs and I/Os		±100	µA	Note 5

NOTES:

1. Parameter applies to the PICCLK and PWRGOOD signals only.
2. VILx,min and VIHx,max only apply when BCLK and PICCLK are stopped. BCLK and PICCLK should be stopped in the low state. See Table 23 for the BCLK voltage range specifications for when BCLK is running. See Table 24 for the PICCLK voltage range specifications for when PICCLK is running.
3. Parameter measured at 10 mA.
4. VCMOSREF and VCLKREF should be created from a stable voltage supply using a voltage divider.
5. (0 ≤ VIN/OUT ≤ VIHx,max).
6. Specified as the minimum amount of current that the output buffer must be able to sink. However, $V_{OL,MAX}$ cannot be guaranteed if this specification is exceeded.

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to:

- *Mobile Pentium® III Processor in BGA2 and Micro-PGA2 Packages 400 MHz, 450 MHz and 500 MHz* datasheet
- *Pentium® III Processor Mobile Module MMC2* datasheet

K1. $I_{CCT, DSLP}$ Specification

In Table 9 of the *Mobile Pentium® III Processor in BGA2 and Micro-PGA2 Packages at 400 MHz, 450 MHz, and 500 MHz* datasheet *(Order Number 24530201), t*he $I_{CCT, DSLP}$ specification was "TBD". It has been removed from the datasheet. The datasheet will be updated in the next revision.